NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

A PICTURE-DESCRIPTOR EXTRACTION PROGRAM
USING SHIP SILHOUETTES

by

CPT Michael J. Bizer

June 1989

Thesis Advisor: Neil C. Rowe

Approved for public release; distribution is unlimited

Unclassified

Security Classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification: UNCLASSIFIED	1b Restrictive Markings
2a Security Classification Authority	3 Distribution Availability of Report: Approved for public release; distribution is unlimited.
2b Declassification/Downgrading Schedule	
4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)
6a Name of Performing Organization: Naval Postgraduate School; 6b Office Symbol (If Applicable): 52	7a Name of Monitoring Organization: Naval Postgraduate School
6c Address (city, state, and ZIP code): Monterey, CA 93943-5000	7b Address (city, state, and ZIP code): Monterey, CA 93943-5000
8a Name of Funding/Sponsoring Organization; 8b Office Symbol (If Applicable)	9 Procurement Instrument Identification Number
8c Address (city, state, and ZIP code)	10 Source of Funding Numbers: Program Element Number; Project No; Task No; Work Unit Accession No

11 Title (Include Security Classification)
A PICTURE-DESCRIPTOR EXTRACTOR PROGRAM USING SHIP SILHOUETTES

12 Personal Author(s)
Bizer, Michael J.

13a Type of Report	13b Time Covered	14 Date of Report (year, month, day)	15 Page Count
Master's Thesis	From To	June 1989	67

16 Supplementary Notation: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17 Cosati Codes		
Field	Group	Subgroup

18 Subject Terms (continue on reverse if necessary and identify by block number)
Artificial Intelligence, Feature Extraction, Boundary Tracing, Ship Recognition, Prolog, Silhouette Identification

19 Abstract (continue on reverse if necessary and identify by block number)
This research examines the practicality of automatically identifying the features of the major structures of ship silhouettes using rule-based extraction and identification techniques. The process was broken into three phases: (a) finding the silhouette boundary, (b) locating the "bumps" (apparent superstructures) on the boundary, and (c) describing the bump features qualitatively using a multidimensional-feature-space classification. The program for the first phase is written in C while the programs in the other two phases are written in MPROLOG and run on a Motorola 68020-based workstation. The programs accurately identified 78% of all bumps examined on six ships. The lists of bump descriptions showed the key differences between two different ships of the same ship-class, indicating future programs could identify ships using the output from the programs of this thesis.

20 Distribution/Availability of Abstract	21 Abstract Security Classification
[X] unclassified/unlimited [ ] same as report [ ] DTIC users	UNCLASSIFIED

22a Name of Responsible Individual	22b Telephone (Include Area code)	22c Office Symbol
Prof. Neil C. Rowe	(408) 646-2462	Code 52Rp

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted — security classification of this page

All other editions are obsolete — Unclassified

Approved for public release; distribution is unlimited.

A Picture-Descriptor Extraction Program using Ship Silhouettes

by

Michael James Bizer
Captain, United States Army
B.S., Western Kentucky University , 1980
M.S., University of Southern California, 1987

Submitted in partial fulfillment of the requirements for
the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1989

ABSTRACT

This research examines the practicality of automatically identifying the features of the major structures of ship silhouettes using rule-based extraction and identification techniques. The process was broken into three phases: (a) finding the silhouette boundary, (b) locating the "bumps" (apparent superstructures) on the boundary, and (c) describing the bump features qualitatively using a multidimensional-feature-space classification. The program for the first phase is written in C while the programs in the other two phases are written in MPROLOG and run on a Motorola 68020-based workstation. The programs accurately identified 78% of all bumps examined on six ships. The lists of bump descriptions showed the key differences between two different ships of the same ship-class, indicating future programs could identify ships using the output from the programs of this thesis.

ACKNOWLEDGEMENTS

I wish to express my deepest appreciation to my wife, Sally, for her encouragement and support during the research and writing of this thesis. Thank-you also to my three daughters for their understanding that "daddy had to go to school a lot."

TABLE OF CONTENTS

I. INTRODUCTION

A. OBJECT RECOGNITION AND IDENTIFICATION

The process of visually identifying an object can be broken into detection, delineation, and identification. Detection locates the object, delineation separates the region the object occupies from the background, and identification determines what the object is. The identification process is of critical importance in military applications. Human analysts remain the only means of performing this task accurately and reliably.

Computers can provide assistance in each of these three areas, although they are weakest in identification. Their processing speed allows for rapid scanning of an area for detection targets. Once detected, there are many ways of extracting the target's shape.

B. MILITARY REQUIREMENTS FOR SHIP IDENTIFICATION

Rapid detection, analysis, and identification of ships is important in military operations. Identification is necessary to determine whether the ship is friendly or potentially hostile. The number of action options available to a commander rapidly decreases the longer it takes to identify the target.

Training humans in ship identification is time-consuming and expensive. Once trained, these skills suffer if not practiced frequently. Additionally, the amount of information to be interpreted can be overwhelming and the time constraints minimal. Computer-assisted identification promises to alleviate these current shortcomings.

Our approach to computer-assisted ship identification is to break a detected object down into its elementary pieces and then identify the pieces. A digitized ship silhouette's boundary is delineated by a threshold comparison. The boundary is converted into a list of line-segment coordinates of contour turn points. Using artificial-intelligence techniques,

these points are sequentially examined to find and extract the coherent pieces of the ship. The size, shape, amount of irregularity, and location of each extracted piece are then examined. Each piece is given an identity of gun-turret/weapons-system, superstructure, weapons-system, mast/support, antenna, radar, or unknown. The list of these pieces and their identity can be reported to humans or could be supplied to future systems similar to [Ref. 1] which could identify the ship from its known pieces and a database search.

C. DESCRIPTION OF THESIS

The remaining chapters of this thesis present background information and describe the identification programs and their results. Chapter II describes some of the current and past research in computer-assisted imagery and object identification. Chapter III examines the problems inherent in object identification and describes areas where computer-assisted systems would improve processing speed and accuracy.

Chapter IV is a discussion of the programs we developed for feature extraction and identification, the algorithms we used, and the output the final program generates. Chapter V will examine the results of the program for different ship images, and chapter VI presents the conclusions and areas for follow-on research.

II. RESEARCH IN COMPUTERIZED OBJECT IDENTIFICATION

A. BACKGROUND

Computer-automated and computer-assisted object recognition and identification has applications in many different areas, including robotics, imagery interpretation, and chemistry [Ref. 2: p.11]. For example, imagery analysts need help in processing the large quantity of digital imagery data they receive. Much of their work involves detecting changes in images taken at different times, a time-consuming task well-suited to computerized scanning [Ref. 3:p. 905]. Programs which enable computers to identify objects have involved conventional procedural image analysis, artificial intelligence (AI), or a combination of both.

B. RESEARCH USING CONVENTIONAL METHODS

1. Shape Recognition Using Binary Images

Grogan [Ref. 4] performed a comparative study of shape recognition and description using binary images. His test objects were six different aircraft arranged in different views. His analysis concentrated on the use of global shape methods to identify and analyze the image boundary. The five methods compared were 1) Fourier descriptors of the boundary, 2) Walsh points of the boundary, 3) the cumulative angular deviant Fourier descriptors, 4) moments of the silhouette, and 5) moments of the boundary.

The research was conducted in 1982 and 1983, and the speed of the programs was good. One of their limitations was the resolution capabilities of the equipment; images were limited to 256 by 256 pixels, so silhouettes were simplified drawings of the aircraft to be identified.

2. Pattern Classification from Elemental Shapes

Research which involved dissecting silhouetted pictorial patterns into specific elemental shapes was conducted by Todd [Ref. 5]. His efforts concentrated on pattern classification. A figure was dissected into subfigures and each subfigure was analyzed according to multiplicity, orientation, position, size, shape and position. The analysis of the subfigures generated a Figure Classification Number (FCN) for the whole pattern, which could be compared against other known FCN's in a database search. A match would allow the figure to be identified.

The subfigures of an object generated by Todd were not each classified as a named feature of the object, but the analysis of these pieces led to an object identification. The program successfully recognized 22 different aircraft silhouettes as airplanes. Each of the airplane's differences were written in their FCN. Thus the result was in a form which only the computer could interpret, unlike the ship image summaries prepared by the ship analysis program presented in this thesis.

3. Identification of Spherical Objects

Computers can rapidly process large amounts of data but have difficulty identifying objects because of the large number of possibilities. Humans skilled in photographic interpretation have problems processing large amounts of imagery data, but are very skilled at identifying an object once it is detected. A more human-like computer identifier was presented by Cox, et al. [Ref. 3], in a detector for all spherical objects contained in an image.

The first step of their detection methodology is parameter extraction, where the position of the light source and image intensity parameters are gathered. The next step is feature detection, using either input from a previous program or by conducting a simple pattern search, which takes about 60 seconds for a 512 by 512 image. The third step is

segmentation, which uses the gradient angle transform to segment an image, creating a candidate region to examine. The final step is validation, in which the region is processed and classified.

Their approach limits the universe of objects the computer must identify to one type while using several different detection algorithms to ensure detection. The approach of this thesis also limits its universe significantly.

4. Parallel Processors and Image Analysis

Because of the large amounts of data a single image contains, research using parallel-processing computers is also being conducted [Ref. 6]. In one example implemented using a Connection Machine, the operator identifies an object for the machine to find and gives an example. The program studies the example and builds a database of knowledge about the object. Using this knowledge, it then examines an image, finding and highlighting any of those objects present in the image. Using the Connection Machine reduces processing time from 30 minutes to under one second for pictures ranging in size from 256 by 256 pixels to 5000 by 5000 pixels [Ref. 7].

This approach works at a lower level of analysis than the approach in this thesis. It could help the analyst in the initial processing stage to highlight areas which should be examined first.

C. RESEARCH USING AI TECHNIQUES

1. Knowledge-Based Airport Identification

One example of research combining conventional and knowledge-based techniques is SPAM, system for airport photographic interpretation using MAPS [Ref. 8]. Using conventional image processing tools, SPAM first detects and labels regions. Then, using rule-based control and recognition methods, it groups major components of airports.

This technique initially suffered from the same limitation mentioned in section B-3, in that its universe of knowledge was very limited, in this case to airports. Later research [Ref. 9] focused on ways to improve the knowledge acquisition process for other universes, resulting in a set of interactive tools. These applications concentrate on identifying all occurrences of a structure, whereas this thesis concentrates on the pieces of an object within a structure.

2. Rule-Based Methods for Image Analysis

Recent work attempted to analyze, segment, and interpret images of printed circuit boards and satellite images of ice flows [Ref. 10]. The methods developed used a combination of procedural and rule-based programming. The process was divided into three levels: low, mid, and high. Low-level processing involved image operations such as thresholding, computation of gradients and non-maxima suppression, etc.

The mid-level processing involved grouping of similar edges and extracting symbolic entities. The mid-level processing extracted the initial symbolic information, including the geometric properties of each linked segment, the forward and backward neighbor codes, and the codes or labels of edges passing through pixels. Further mid-level processing joined small segments and edges together.

The high-level processing attempted to identify the extracted entity, deriving a symbol that has meaning to either humans or another follow-on process. This step used a hypothesis generation/reduction/verification model. The goal was to produce a symbolic description of the input image. For a circuit board, this would be a description of the length, location, and orientation of segments.

The program works towards the same goal as ours in trying to take a low-level group of data and bring the description of the object to a symbolic level understood by humans. The final symbolic outputs of the programs are different. Theirs prints the

location and size of an object, while ours goes a step further and identifies what type of object is being examined.

3. Recognizing Man-Made Objects in Aerial Images

Another system for analyzing aerial imagery was developed entirely with AI languages, specifically Lisp and ART (the Automated Reasoning Tool) [Ref. 11]. The approach differed from [Ref. 10] in that it was goal-driven: the system identified all requested object types in an image.

The application was tested with airport scenes and also as a target-cueing aid for FLIR (Forward-Looking Infrared) imagery. For low-level processing, the system could utilize several different image-processing techniques. The program could select the proper technique based on parameters such as the application area, sensors used, resolution, weather conditions, and quality of data. The low-level processing concentrated on edge detection, region segmentation, and other image enhancement functions.

Intermediate-level processing was responsible for creating and filling slots for each region. These slots included coordinates, area, variance, connectivity, compactness, and other descriptions.

Knowledge about the area of interest improved the interpretation process. The system included information on basic terrain features such as power plants, roads, railroads, rivers, forests, etc. Based on the goal given the system, it would also generate a dynamic model for the goal objects. Then the high-level processing would classify regions into goal objects, using the information in the region slots while attempting to make the best matches possible.

The approach provides larger flexibility than the one presented in this thesis. Also, instead of identification, it concentrates on detecting and highlighting objects for the human operator to examine.

III. METHODS OF SHIP IDENTIFICATION

For people to develop the necessary expertise in ship identification is a time-consuming and personnel-intensive task. There are hundreds of types of ships to be studied, each with its own set of identifying characteristics. Learning these characteristics requires days of study, as well as frequent refresher training to maintain the acquired skills.

Instruction methods for image interpretation generally follow a rule-based approach. The student must develop his working knowledge of ships and then memorize key differences between types of ships. The first step is to teach the student the different features of a ship. These features usually stand out from a silhouette or image and can be peeled from the ship and analyzed independently. During this analysis the feature's identity is further refined to as much detail as possible. This is shown in the table below.

TABLE 3.1. Example Identifications of Ship Structures

Feature	Intermediate Identity	Exact Identity
gun-turret	1 5-in gun tube, located forward, square turret	Mark 54
weapons-system	square launcher, forward, several tubes	ASROC Launcher
radar	large, square antenna, probably air-search.	Lockheed SPS-40
large structure aft	box-like shape, clear area beside, several antennas on box	Hangar for LAMPS

Let us assume the student is learning about air-search radar. The student must first learn a series of broad rules for each feature, like that air-search radar usually have large, squared antennas, rotate 360 degrees, and are located high in the ship superstructure. Partial classification (such as recognition that the feature is a radar) of a feature can allow

the ship identification process to still proceed. The next step is learning to distinguish between different air-search radar. For example, the Lockheed SPS-40 radar antenna has a triangular base, a square antenna with tapered ends, and a feed horn that hangs over the top of the antenna.

Once a particular radar has been accurately identified, the student must learn to associate it with particular ships. For example, the SPS-40 radar is found on several US FFG class ships, including the BRONSTEIN and GLOVER classes. The student then tries to learn every ship-class known to use the SPS-40 radar. Such detailed knowledge deteriorates rapidly without frequent use or refresher training.

Once the student is capable of identifying the different features, he next learns the rules for identifying particular ships. This requires learning the differences that the same features may have on different ships. These differences include the way different features are combined as well as where on the ship these features are located. The learning process is again rule-based, as in the GARCIA-class FFG's which have one Mk-54 gun-turret forward and one Mk-54 gun-turret aft, an ASROC launcher behind the forward turret, a Lockheed SPS-40 radar, a single stack, and an SPS-10 radar. The difference between the SAMPLE and BRADLEY, both GARCIA-class FFG's, is the BRADLEY is fitted for the LAMPS helicopter and has a hanger on the rear deck. The rules are learned for all classes of ships, and the skills will rapidly deteriorate without constant use.

IV. A PICTURE-DESCRIPTOR EXTRACTION PROGRAM

A. OVERVIEW

Our feature extraction and identification process involves three stages; (1) finding the silhouette boundary (outline), (2) locating the bumps on the silhouette, and (3) extracting and identifying the features of the bumps. Stage 1 is implemented in C and stages 2 and 3 are written in MPROLOG. The program flow is shown in Figure 4.1 below. This chapter is broken into three corresponding sections.

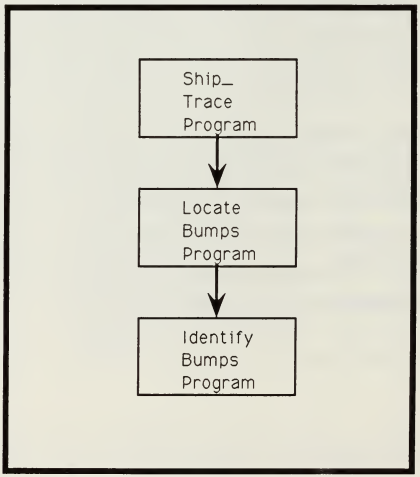


Figure 4.1. Diagram of Program Modules

Figure 4.2 contains a labeled ship silhouette, demonstrating some bump identifications our program attempts to make.

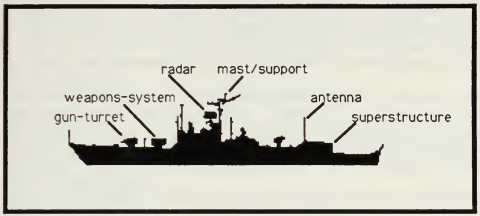


Figure 4.2. Silhouette with Labeled Bumps

B. FINDING THE SILHOUETTE BOUNDARY

The ship silhouettes are digitized using an Eikonix digitizer and SUN Workstation. The digitizing program is **scanit**, written by David S. Hill for a computer graphics class. The ship must be positioned upright and level with the bow on the left side. The orientation is important as the second stage exploits the identity of y-coordinates. Using an integration value of about 12,000, the digitized image is captured into a file as a 1000 by 1000 array of the pixel values. This file is transferred to the VAX 11/785 computer for boundary processing.

The ship silhouettes used were taken from Jane's All the World's Fighting Ships [Ref. 12]. Six different ships of the frigate class were used. The selection criteria for the silhouettes were they have many mutual features in common, with differences in the size and location of the features. Each digitized silhouette's file is processed separately by the **ship_trace** program, with the resulting files of line segments passed on to the next stage.

The first problem to be solved when finding the boundary of the digitized image is selecting a proper threshold. The ship silhouettes were solid black shapes on white paper. Although black would be indicated by a pixel value near 0 (0 hex) and white would be near 255 (ff hex), setting the threshold to catch the change from black to white requires some adjustment for each picture. This was necessary to ensure that any noise captured with the image was tuned out. Also, a threshold too close to black would tend to fill in many of the fine details of the superstructure, while a threshold too close to white would allow breaks in antennas or cracks through the ship to appear in the image. To help in selecting the proper threshold value, a small program was written which printed out a portion of the silhouette with asterisks for the dark areas and spaces for the light ones. By running this program using different values and viewing the results, the threshold value could be fine-tuned to capture the right amount of detail.

Once the threshold value is selected, the boundary-finding program can be run. This program was designed to walk along the boundary[1], of the silhouette, recording (x,y) coordinates of points whenever it changed direction. The program uses the four principal directions, up, down, left, and right. Its goal was to start at the left end and keep moving toward the right end of the ship. The approach is similar to Pavlidis' contour tracing algorithm [Ref 13:p. 143]. The program in effect traces the outline of the ship, creating a file of turn points. Each pair of points denotes a line segment. A long straight antenna would be converted into two or three points, while a gun turret might create fifty or more points.

---

[1]The boundary is located along the black cells delineating the top of the ship. In effect, the program would walk the deck of the ship, recording locations where it changed direction. The locations were expressed in the coordinates of the black cells.

Originally, we intended to trace the outlines using eight directions, the four explained above as well as the diagonals. But diagonal lines can also be represented by a pattern of horizontal and vertical moves. Although this creates many more turn points, finding and recording diagonals requires looking at a larger group of points during each iteration than our algorithm was designed for.

Turn points found are written to a data file. This data file is then converted to an acceptable MPROLOG format: a list with each point in the list being represented as cp(<x-coordinate>,<y-coordinate>).

C. LOCATING THE BUMPS

The extraction algorithm was designed to look for matching pairs of up-turns and return-to-horizontals. These are the definers of a single "bump", the point where it starts going up and the point where it returns to the horizontal at a height equal to that of the up-turn. By detecting those two points, and capturing all the line segments between them, a bump can be extracted. A modified algorithm is used to find bumps which start and end at different heights. The program repeatedly examines the point where it currently is, the previous point, and the next point. From these, it determines if the middle point was located at an up-turn. If so, it saves the current location coordinates in a list of up-turns. This is repeated for every successive triple of points.

In order to capture only true corners, an up-turn was required to have at least a three-pixel vertical movement. This helps ignore parts of the silhouette which were jagged due to noise. The return-to-horizontal which matched these up-turns was only required to have a one-pixel movement. By having the up-turn constraint select only well-defined corners, the return-to-horizontal constraint could be looser to make it easier to find the matching corner of the bump. Figure 4.3 shows examples.

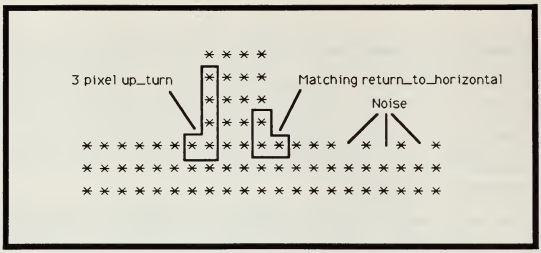


Figure 4.3. Example of Normal Bump and Image Noise

Not all bumps on a ship can be caught this way, so two variations of the algorithm are required. One operates as an inverse of the first. If the return-to-horizontal has a three-pixel or more vertical movement preceding it, the program turns around and retraces the bumps until it finds a one-pixel or two-pixel vertical movement where the y coordinates are equal to the return-to-horizontal's. An example is shown in Figure 4.4.

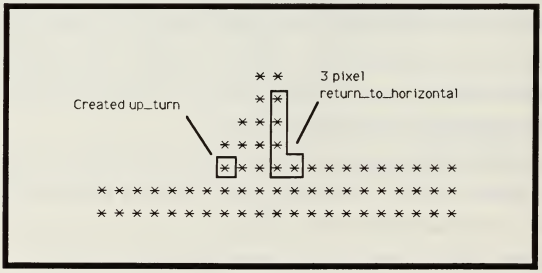


Figure 4.4. Example of Created Up-turn

The final variation is designed to detect and create a bump when a vertical movement passes below a previously labeled up-turn. That terminates all unmatched bumps whose up-turns were in the vertical-movement height range. An example is shown in Figure 4.5.

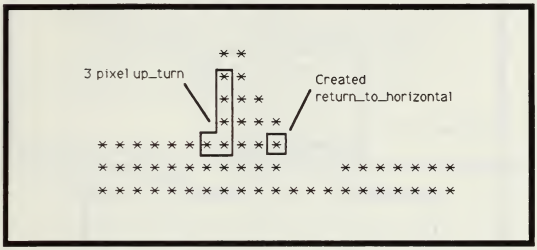


Figure 4.5. Example of Created Return_to_Horizontal

The file containing matching pairs of up-turns and return-to-horizontals is kept in MPROLOG format. When the algorithm reaches the end of the ship, it closes off any bumps still in the up-turn list and closes the file.

D. BUMP FEATURE IDENTIFICATION

1. Removing the Shape from the Silhouette

The shape analysis program is written in MPROLOG. It iterates through a list of bumps, finding their size, length-to-width ratio, curviness, and location. These descriptors are then assigned categories and rules for classification of bumps from descriptor categories are applied.

To do this, the program traces the outline of the bump using the list of turn points. If the bump has other bumps within it, these bumps are stripped off so only the basic shape

remains. If a bump within the bump is removed, the program creates a straight line segment where the bump was. This is shown in Figure 4.6.

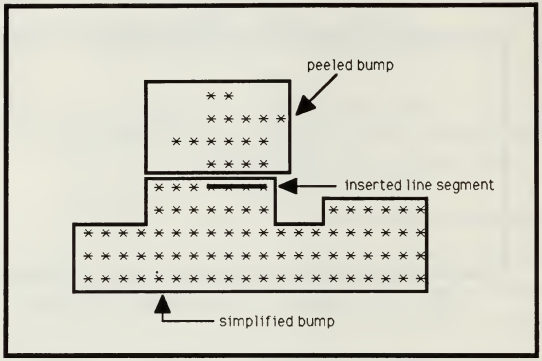


Figure 4.6. Example of Peeling Bump and Creating Line Segment

Next, each bump is analyzed as to curviness, horizontal location, length-to-width ratio, orientation, and size. The number of turn points in the bump list roughly indicates the curviness of the shape (See Figure 4.7). The location of the shape is determined as bow, forward, mid_ship, aft, or stern; this is useful as gun systems tend to be forward or aft on ships, while radar systems are usually found in the middle third of the ship, around the superstructure. The length-to-width ratio of the object is found from the minimum and maximum x and y values, which create an imaginary box around the object, as exemplified in Figure 4.8. Particular length-to-width ratios can be described as the shapes square, pole, rectangle, etc. Box orientation (flat_rectangle, tall_rectangle) can also be described. The box size in relation to the ship is also used.

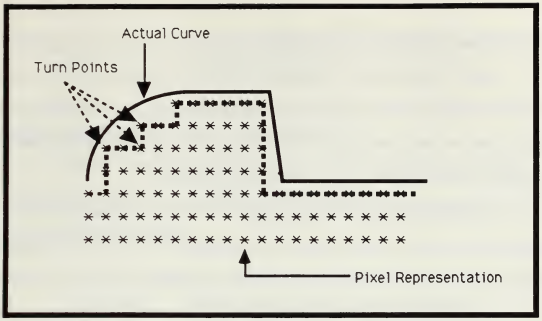


Figure 4.7. Example of Curve Approximation

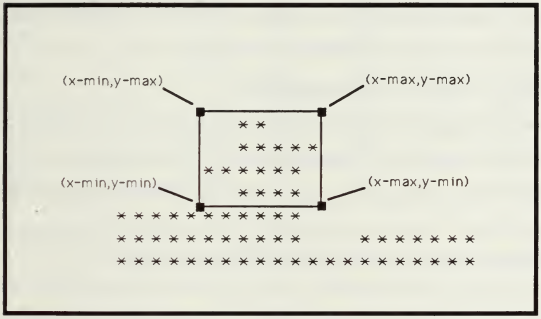


Figure 4.8. Example of Box Surrounding a Shape

2. Shape Analysis and Identification

After all descriptors of a bump have been found, shape rules (for antenna, radar, gun-turret/weapons-system, weapons-system,mast/support,or superstructure) are applied. For example, the descriptors [pole, tall_rectangle, medium, mid_ships] indicate an antenna of some type, and "antenna" is put into the identification list.

Sometimes a bump will match more than one rule. For example, a shape may have attributes of a radar as well as those of a superstructure. Then both rule descriptions are added to the list and the word "or" is inserted between them.

Sometimes a bump will not match any of the rules. This may be due to noise introduced when the image was digitized, or the bump might be an uncommon shape for these types of ships. In this case, the word "unknown" is inserted in the identity list. Since the location of the shape and its description list is also kept for future use, a more intelligent program might later be able to identify the shape.

E. FINAL OUTPUT OF THE PROGRAM

When the programs have completed all processing, the final file consists of a listing of all the extracted shapes, their descriptors, and their identity, if known. The shapes are identified by their starting and ending coordinates, their descriptor list, and their identification list. This file is then available for processing by a future program which could match a ship class to the descriptors our program has created.

V. RESULTS OF PICTURE-DESCRIPTOR EXTRACTION PROGRAM

A. FINDING THE SILHOUETTE BOUNDARY

The performance of the first-stage program is shown in Table 5.1 below. The measurements indicate the size of the MPROLOG file produced, the number of turn points in the file, and the threshold setting.

TABLE 5.1. Outline Extraction Program Results

Ship Name	MPROLOG File Size	Number of Points	Threshold
GARCIA (LAMPS)	12.5 Kb	962	60
GLOVER	17.9 Kb	1,380	50
BROOKE	12.4 Kb	952	50
BRONSTEIN	8.1 Kb	634	50
KNOX	10.4 Kb	798	55
GARCIA	11.9 Kb	913	50

As the table shows, the resulting MPROLOG file is not very large, since each of the original digitized image files was 990 Kb in size. The only problem encountered involved the silhouette of the BRONSTEIN: its digitized image had more random noise than the others, and the program required several runs to find the bow of the ship, the starting point.

B. LOCATING THE BUMPS

The second stage of finding and marking the location of bumps on the silhouettes was the slowest stage. The program could process any ship except the GLOVER in one run (the number of turn points in the GLOVER's data file had to be split in half to allow the program to run, for otherwise the program crashed after reading in the data points and the required program modules). A listing of the CPU time used is shown in Table 5.2.

TABLE 5.2. Execution Time for Bump Locating Program

Ship Name	CPU Time (minutes:seconds)
GARCIA (LAMPS)	31:07
GLOVER	39:11
BROOKE	30:36
BRONSTEIN	13:17
KNOX	21:37
GARCIA	27:47

Table 5.3 summarizes the number of bumps found.

TABLE 5.3. Results of the Bump Locating Program		
Ship Name	Image Length (Pixels)	Number of Bumps
GARCIA (LAMPS)	735	37
GLOVER	933	48
BROOKE	769	44
BRONSTEIN	700	34
KNOX	757	37
GARCIA	758	26

To test the program with a different size, the GLOVER silhouette was digitized so that it completely filled the imaging window, adding more detail to the image than the others, and this silhouette generated the largest number of bumps. The smallest number of bumps was generated by the GARCIA-class FFG without a LAMPS silhouette because the rear of the ship has few.

C. BUMP FEATURE IDENTIFICATION

Table 5.4 summarizes the results of the third program in the identification process. A successful identification is when a bump is identified either as itself or as a part of an or'd list. If the bump is classified as **unknown**, it is not counted as a successful identification. The use of unknown as a descriptor prevents the program from a wrong identification, i.e. identifying a radar as a gun-turret.

TABLE 5.4. Statistics of Bump Identification Program				
Ship Name	Number Identified	Number Unknown	Number Not Done	Percent Successful
GARCIA (LAMPS)	32	4	1	86%
GLOVER	40	8	0	83%
BROOKE	38	6	6	86%
BRONSTEIN	26	6	6	76%
KNOX	27	10	0	73%
GARCIA	14	6	6	53%
TOTALS	177	42	7	78%

As the table shows, the identification program is successful in identifying a bump about three of four tries. The number of bumps a ship has to extract does not seem to affect the program's success rate. A bump falling in the **unknown** category is not necessarily a failure; it may have an unusual shape, there may be noise in the image that has distorted its shape, or the detail from the silhouette may not provide enough information to make a definitive identification. Some features are more helpful in the identification process than others. For frigate-class ships, the key features are the number, location, and type of weapons systems, the number of radar and antennas, and the presence or absence of certain superstructures like helicopter hangars.

To illustrate bump features, the differences between a GARCIA-class FFG and the GARCIA FFG with LAMPS can be seen in Figure 5.1.

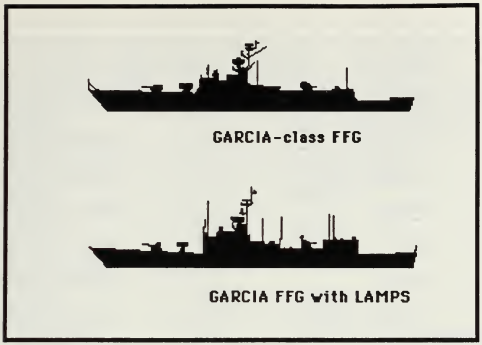


Figure 5.1. Silhouettes of GARCIA-class FFGs

The ships have the same basic structure and similar weapons systems and radar. The key differences are the number of antennas and the presence of the helicopter hangar on the rear deck of the FFG with LAMPS. Table 5.5 describes a GARCIA-class FFG without the LAMPS helicopter modification. Comparing it and Table 5.6 illustrates the differences our program detects between two similar ship silhouettes.

TABLE 5.5. Features for GARCIA-class FFG (no LAMPS)	
Feature on the Ship	**Feature Identified by Program**
gun-turret	weapons-system
weapons-system	weapons-system
radar	radar or superstructure
radar	radar or superstructure
radar	(program cannot process)
weapons-system	weapons-system
antenna	antenna

TABLE 5.6. Features for GARCIA FFG with LAMPS	
Feature on the Ship	**Feature Identified by Program**
gun-turret	gun-turret/weapons-system
weapons-system	weapons-system
antenna	antenna
antenna	antenna
radar and mast	mast/support
antenna	antenna
mast/support	mast/support or superstructure
antenna	antenna
antenna	antenna
gun-turret	gun-turret/weapons-system
antenna	antenna
antenna	antenna
hangar superstructure	superstructure

D. PROGRAM LIMITATIONS DUE TO HARDWARE CONSTRAINTS

A ship's bumps could be identified in two to five runs of the program. The requirement for multiple runs was due to hardware and software limitations of the ISI Workstations. The MPROLOG statement table does not use any garbage-collection techniques to free up memory after it is no longer needed. Because of the large amount of recursive list processing used in the extraction process, the statement table fills up quickly and the program halts. After moving the coordinates of bumps that have been processed to

the end of the list of bumps, the program can be restarted. It will continue to process the remaining bumps, adding their descriptions as they are developed.

A similar limitation experienced by the feature extraction and identification program was its inability to process seven bumps when analyzing the two GARCIA-class frigates. The error occurs as the program attempts to strip off several bumps mounted on top of the ships' central superstructure. The number of points to be extracted for the bump and the number of points being processed to peel off a bump are too many for the system stack. However, if the feature causing the problem is removed from the list and the program is restarted[2], the program operates normally. The effects of this inability to process six bumps is shown in the shorter length of Table 5.5 and the overall lower identification success rate for the GARCIA-class FFG without LAMPS. The problem can be solved either by adding more memory or changing the software.

---

[2] In this case, the ordering of the list of bumps is changed, either moving the problem bumps behind the end of list flag or by removing them from the list. The program is then restarted from the beginning, unlike the restarts mentioned earlier where the program picks up where it left off.

VI. CONCLUSION

A. SUCCESS OF FEATURE EXTRACTION AND IDENTIFICATION

We have demonstrated that the key features from a ship silhouette can be extracted, analyzed, and identified to produce a list describing and naming the features. The results we have seen are similar to other research using knowledge-based and artificial-intelligence techniques. The average success rate for six frigate-class ships was 78%, with values ranging from 53% to 86%.

These programs provide a versatile front-end method for automated ship recognition and identification. The description and identification information contained in the final listing could be used by other programs to identify the type of ship or possibly the name of the ship. Such a program, combined with an intelligent tutor program, would relieve instructors of the administrative and repetitive tasks of updating silhouette libraries.

The MPROLOG programs suffer from hardware and software limitations. The amount of information contained in a silhouette is not that large, but the recursive analysis used by the programs quickly use up available memory and stack space. Many other Prolog dialects should not have these limitations.

B. AREAS FOR FUTURE WORK

The program could be extended to handle all ships, regardless of nationality or type. The program could also be improved by adding the ability to detect and analyze objects with diagonal appendages. Our program converts diagonals into horizontal and vertical steps. Extending it to interpret diagonals would allow the program to better pick apart pieces of the superstructure and identify more of the antenna and mast detail.

Adding the ability to reason from known bumps would allow the program to go back and reevaluate bumps previously identified as unknown and make an identification. Identification is possible by knowing what ship classes partially match the current silhouette and looking for unknown bumps that could match known ship structures in the ship classes being examined.

These programs could be put together with an intelligent tutoring program. The combined system would be a useful tool for developing and maintaining-silhouette identification skills.

APPENDIX A - C PROGRAM SOURCE CODE

SHIP_TRACE.C

This program finds the bow and stern of the digitized image and creates a binary file **line.coords** is then used by program **format.c** to create a MPROLOG data file of the turn points.

```
#include <stdio.h>
main()
{
    char    PTS [250][1000];
    char    row_data[1000];
    int     num_pixels[1000];
    int     datafile,pfile,pfile1;
    int     XCOORD,YCOORD;
    char    code;
    int     X,Y;
    int     row,i;
    int     fd[1];
    int     info[2];
    short   TYPE,ROWS,COLUMNS;
    int     max_rows, max_columns, max_PTS;
    int     xstartpoint, ystartpoint;
    int     xendpoint, yendpoint;
    int     threshold;                  /* adjust to picture quality */

/*  open datafile  */
printf("starting program \n");
if ((datafile = open("/scratch/bizer/pic.brooke",0)) < 0 )
  {
    printf("can't find it \n");
    exit();
  }

/*   open output file  */
if ((pfile = creat("line.coords",0744)) < 0 )
  {
    printf("can't open output line.coords\n");
    exit();
  }

/*   open output file  */
if ((pfile1 = creat("line.info",0744)) < 0 )
  {
    printf("can't open output line.info \n");
    exit();
```

```
  }
/*  pull off header information  */

      read(datafile,&TYPE,2);
      read(datafile,&ROWS,2);
      read(datafile,&COLUMNS,2);
      printf("type %d max_rows %d max_columns %d \n",TYPE,ROWS,COLUMNS);
      max_rows=1000;
      max_columns = 1000;

/*clean off empty space above*/
for (row = 0; row < 280; row++)
      read(datafile, &row_data[0], max_columns);
printf("cleaned off blank space \n");

/*   get a row of information   */
for (row = 280; row < 510; row++)
      read(datafile, &PTS[row-280][0], max_columns);

printf(" Array is initialized.\n");

/*********************************************************************/
/*  Initialize tracing program                                       */
xstartpoint = 1000;
xendpoint = 0;

for (row = 0; row < 220; row ++)
  for (i = 20; i < max_columns; i++)
  {
     if (PTS[row][i] < 55 && i < xstartpoint)
     {
       xstartpoint = i;
       ystartpoint = row;
     }

     if (PTS[row][i] < 55 && i > xendpoint)
     {
       xendpoint = i;
       yendpoint = row;
     }
  }

XCOORD = xstartpoint;
YCOORD = ystartpoint;
X = XCOORD;
Y = YCOORD + 280;
ystartpoint = ystartpoint + 280;
yendpoint = yendpoint + 280;

printf("XStartpoint = %d  XEndpoint = %d\n",xstartpoint,xendpoint);
printf("YStartpoint = %d  YEndpoint = %d\n",ystartpoint,yendpoint);
```

```
write(pfile1,&ystartpoint,4);
write(pfile1,&xstartpoint,4);
write(pfile1,&yendpoint,4);
write(pfile1,&xendpoint,4);
printf("made it past writing to line.info \n");

code = 'R';
prev_code = 'Z';
threshold = 55;
while (XCOORD < xendpoint)
{
  while (code == 'R')
  {
   if (PTS[YCOORD-1][XCOORD] < threshold)
      {
      printf("entering with code = R\n");
      write(pfile,&Y,4);
      write(pfile,&X,4);
      YCOORD =  YCOORD - 1;
      Y = Y + 1;
      code =  'U';
      }
   else if (PTS[YCOORD][XCOORD+1] < threshold)
      {
      X = X + 1;
      XCOORD =  XCOORD + 1;
      }
   else if (PTS[YCOORD+1][XCOORD] < threshold)
      {
      write(pfile,&Y,4);
      write(pfile,&X,4);
      YCOORD =  YCOORD + 1;
      Y = Y - 1;
      code =  'D';
      }
   else if (PTS[YCOORD][XCOORD-1] < threshold)
      {
printf("entering with code = R\n");
      write(pfile,&Y,4);
      write(pfile,&X,4);
      X = X - 1;
      XCOORD =  XCOORD - 1;
      code =  'L';
      }
  }
  while (code == 'D')
  {
   if (PTS[YCOORD][XCOORD+1] < threshold)
      {
      write(pfile,&Y,4);
```

```
    write(pfile,&X,4);
    XCOORD = XCOORD + 1;
    X = X + 1;
    code = 'R';
    }
 else if (PTS[YCOORD+1][XCOORD] < threshold)
    {
    YCOORD = YCOORD + 1;
    Y = Y - 1;
    }
 else if (PTS[YCOORD][XCOORD-1] < threshold)
    {
    write(pfile,&Y,4);
    write(pfile,&X,4);
    X = X - 1;
    XCOORD = XCOORD - 1;
    code = 'L';
    }
 else if (PTS[YCOORD-1][XCOORD] < threshold)
    {
    write(pfile,&Y,4);
    write(pfile,&X,4);
    YCOORD = YCOORD - 1;
    Y = Y + 1;
    code = 'U';
    }
}
while (code == 'L')
{
 if (PTS[YCOORD+1][XCOORD] < threshold)
    {
    write(pfile,&Y,4);
    write(pfile,&X,4);
    YCOORD = YCOORD + 1;
    Y = Y - 1;
    code = 'D';
    }
 else if (PTS[YCOORD][XCOORD-1] < threshold)
    {
    X = X - 1;
    XCOORD = XCOORD - 1;
    }
 else if (PTS[YCOORD-1][XCOORD] < threshold)
    {
    write(pfile,&Y,4);
    write(pfile,&X,4);
    YCOORD = YCOORD - 1;
    Y = Y + 1;
    code = 'U';
    }
```

```
    else if (PTS[YCOORD][XCOORD+1] < threshold)
        {
        write(pfile,&Y,4);
        write(pfile,&X,4);
        X = X + 1;
        XCOORD = XCOORD + 1;
        code = 'R';
        }
    }
    while (code == 'U')
    {
     if (PTS[YCOORD][XCOORD-1] < threshold)
        {
        write(pfile,&Y,4);
        write(pfile,&X,4);
        X = X - 1;
        XCOORD = XCOORD - 1;
        code = 'L';
        }
     else if (PTS[YCOORD-1][XCOORD] < threshold)
        {
        YCOORD = YCOORD - 1;
        Y = Y + 1;
        }
     else if (PTS[YCOORD][XCOORD+1] < threshold)
        {
        write(pfile,&Y,4);
        write(pfile,&X,4);
        XCOORD = XCOORD + 1;
        X = X + 1;
        code = 'R';
        }
    else if (PTS[YCOORD+1][XCOORD] < threshold)
        {
        write(pfile,&Y,4);
        write(pfile,&X,4);
        YCOORD = YCOORD + 1;
        Y = Y - 1;
        code = 'D';
        }
    }
   } /*  end while X < endpt  */

close(datafile);
close(pfile);
close(pfile1);

}/*  end pgm  */
```

FORMAT.C

This program reads **line.coords**, converts the turn points to MPROLOG format and writes the points to file **coords.pro.**

```
#include <stdio.h>
#include <sys/file.h>

main()
{
    FILE    *datafile, *datafile1, *pfile;
    int     row;
    int     i;
    int     xcoord, ycoord;
    int     max_rows;
    int     testvalue;
    int     first;

    /* open datafile  */

    if ((datafile = fopen("line.info", "r")) == NULL) {
        printf("can't find it info\n");
        exit();
    }
    printf("opened line.info\n");

    if ((datafile1 = fopen("line.coords", "r")) == NULL) {
        printf("can't find it coords \n");
        exit();
        }
    printf("opened line.coords\n");

    /** open output file   **/
    if ((pfile = fopen("coords.pro","w")) == NULL ) {
        printf("can't open output \n"); exit();
        }
    printf("opened output file\n");

    fprintf(pfile,"module coords.\n");
    fprintf(pfile,"/*$eject*/\n ");
    fprintf(pfile,"body.\n");
    fprintf(pfile,"\n ");

    /* pull off starting and ending coordinates */
    /* coords are read as Y,X and written as X,Y */

        printf("before fscan \n");
        fread(&ycoord, sizeof(int),1,datafile);
        fread(&xcoord, sizeof(int),1,datafile);
```

```
        fprintf(pfile,"start_pt([%d,%d]).\n",xcoord,ycoord);
        fread(&ycoord, sizeof(int),1,datafile);
        fread(&xcoord, sizeof(int),1,datafile);
        fprintf(pfile,"end_pt([%d,%d]).\n",xcoord,ycoord);
    fprintf(pfile,"\npoints([");
    printf("starting to process rows\n");

    /* read a row of data and write back out in MPROLOG format */
    for (first=1;first < 1999;first++) {
        if (fread(&ycoord, sizeof(int),1,datafile1) == 0)
                break;
        if (fread(&xcoord, sizeof(int),1,datafile1) == 0)
                break;
        if (first != 1)
                fprintf(pfile,",\n");
        fprintf(pfile, "cp(%d,%d)", xcoord, ycoord);
    }
    fprintf(pfile,"]).\n");
    fprintf(pfile,"endmod /* coords.pro */\n");
    fclose(datafile);
    fclose(datafile1);
    fclose(pfile);
}
```

APPENDIX B - MPROLOG PROGRAM SOURCE CODE

PROCESS.PRO

This program examines the turn points from **coords.pro** and locates all bumps on the silhouette.

```
module process.
/*$eject*/
body.
import(add_statement/1, del_statement/1).

dynamic (coord_list / 1).
dynamic (cp / 2).
dynamic (pts / 1).
dynamic (index_counter / 1).
dynamic (up_turn / 1).
dynamic (bump / 4).

total_length(L) :-
        start_pt([X,Y]), end_pt([X1,Y1]), L is X1-X .


set_up(L) :- points(L),
        asserta(pts(L)),
        stars,
        write('START_POINT -->'),start_pt([X,Y]),write(X),write(' '),write(Y),
        write('END_POINT -->'),end_pt([X1,Y1]),write(X1),write(' '),write(Y1),
        nl, stars,
        open(1,"bumps.pro"),
        tell(1,"bumps.pro"),
        write('module bumps.'),nl,
        write('/*$eject*/'),nl,
        write('body.'),nl,nl,
        write('bumps(['),
        told(1).

go :-  state(system_time,START),
        add_statement(up_turn([])),
        stars,
        set_up(L),
        run(X,Y),
        display_stats,
        state(system_time,FINISH),
        REAL_TIME is FINISH - START,
        write("Actual Time = "), write(REAL_TIME),
        write(" shown as HHMMSS").
```

```
run(X,Y) :- points(L),index1(L,0,PP,P,PN),
        get_pts(PP,P,PN,XP,YP,X,Y,XN,YN),
        walk(XP,YP,X,Y,XN,YN),add_statement(index_counter(0)),
        iterate(run1),
        close_bump_file.

run1 :- points(L),index_counter(K),K2 is K+1,      /* Iterates this pred */
        index1(L,K2,PP,P,PN),                      /*  until runs out  */
        get_pts(PP,P,PN,XP,YP,X,Y,XN,YN),          /*  of coordinates  */
        walk(XP,YP,X,Y,XN,YN),                     /*  to process.     */
        del_statement(index_counter(K)),
        add_statement(index_counter(K2)),
        display_stats,!.

done :- points(L),length(L,N),index_counter(K), K>=N-3.

display_stats :- nl,stars,
        state(main_stack,[U,C]),
        write("main_stack used = "),write(U),nl,
        state(statement_table,[U1,C1]),
        write("statement_table used ="),
        write(U1),write(" "),write(C1),nl,
        state(cpu_time,TF),
        stars,write("cpu time = "),write(TF),nl,
        stars.

get_pts(PP,P,PN,XP,YP,X,Y,XN,YN) :-          /* Get three points to process */
                PP =.. [cp,XP,YP],
                P =.. [cp,X,Y],
                PN =.. [cp,XN,YN].


walk(XP,YP,X,Y,XN,YN) :-
                change_up(XP,YP,X,Y,XN,YN).
walk(XP,YP,X,Y,XN,YN) :-
                below_upturn(XP,YP,X,Y,XN,YN).
walk(XP,YP,X,Y,XN,YN) :-
                rtn_horizontal(XP,YP,X,Y,XN,YN).
walk(XP,YP,X,Y,XN,YN) :-
                create_upturn(XP,YP,X,Y,XN,YN).
walk(XP,YP,X,Y,XN,YN) :- !.

change_up(XP,YP,X,Y,XN,YN) :- XP<X,          /* record an up_turn.      */
        L is YN - Y,
        L>2,
        up_turn(UL),append([(X,Y)],UL,UL1),
        add_statement(up_turn(UL1)),
        del_statement(up_turn(UL)),!.
```

```
rtn_horizontal(XP,YP,X,Y,XN,YN) :-          /* bump rtn_to_horiz at same  */
        YP>Y                                /*  level it started, so      */
        Y=YN,                               /*  write coordinates.        */
        X < XN,
        up_turn(UL),
        member((XU,Y),UL),
        delete((XU,Y),UL,UL1),
        del_statement(up_turn(UL)),
        add_statement(up_turn(UL1)),
        stars,
        write_to_screen(XU,Y,X,Y),
        write_to_file(XU,Y,X,Y),nl,!.

below_upturn(XP,YP,X,Y,XN,YN) :-            /* rtn_to_horiz is below latest*/
        up_turn(UL),YP>Y,Y=YN,              /*  up_turn, so create the    */
        member((XU,YU),UL),                 /*  rtn_to_horizontal coord.  */
        YU>Y,X>=XU,
        add_statement(bump(XU,YU,X,YU)),
        delete((XU,YU),UL,UL1),
        add_statement(up_turn(UL1)),
        del_statement(up_turn(UL)),
        write_to_screen(XU,YU,X,YU),stars,
        repeat,check_duplicate(XP,YP,X,Y,XN,YN),
        write_to_file(XU,YU,X,YU),!.

create_upturn(XP,YP,X,Y,XN,YN) :-
        Y + 3 < YP                          /* rtn_to_horiz is below latest */
        Y = YN,                             /*  up_turn, so create        */
        X + 2 <= XN,                        /*  the upturn.               */
        up_turn(UL),
        find_near_up_turn(UL,Y,FX,FY),
        COORD =.. [cp,FX,FY],
        points(L),
        locate_coords(COORD,L,LL),
        find_crossing(LL,Y,XCOORD),
        write_to_file(XCOORD,Y,X,Y),
        write_to_screen(XCOORD,Y,X,Y).


check_duplicate(XP,YP,X,Y,XN,YN) :-         /* Make sure the rtn_to_horiz */
        below_upturn(XP,YP,X,Y,XN,YN),!.    /*  doesn't include multiple  */
check_duplicate(XP,YP,X,Y,XN,YN).           /*  bumps.                    */

find_near_up_turn([(FX,FY)|RL],Y,FX,FY) :-
        FY < Y, !.
find_near_up_turn([A|RL],Y,FX,FY) :-
        find_near_up_turn(RL,Y,FX,FY).
```

```
find_crossing([A,B|LL],Y,XCOORD) :-                    /* Returns vertical coord    */
        A =.. [cp,AX,AY], AY < Y,                      /*   where up_turns Y coords */
        B =.. [cp,XCOORD,BY], BY >= Y,!.               /*   was crossed.            */
find_crossing([A,B|LL],Y,XCOORD) :-
        find_crossing([B|LL],Y,XCOORD).

locate_coords(COORD,[A|L],[A|L]) :-                    /* Find a coordinate from list*/
        first_one(COORD,[A|L]),!.
locate_coords(COORD,[A|L],LL) :-
        locate_coords(COORD,L,LL).

first_one(X,[X|L]).

write_to_screen(X,Y,X1,Y1) :-
        stars,
        write('bump at '),write(X),write(' '),write(Y),write(' '),
        write(X1),write(' '),write(Y1),nl.

write_to_file(X,Y,X1,Y1) :-
        tell(1,"bumps.pro"),
        write('b('),
        write(X),write(','),
        write(Y),write(','),
        write(X1),write(','),
        write(Y1),write('),'),nl,
        told(1).

close_bump_file :-
        up_turn(UL),
        close_up_turns(UL),
        tell(1,"bumps.pro"),
        write('b(999,999,999,999)]).'),nl,
        write('endmod /*bumps.pro */'),
        told(1).

close_up_turns([]).                                    /* Closes any up_turns left  */
close_up_turns([(X,Y)|L]) :-                           /*   in the list.            */
        end_pt([XE,YE]),write_to_file(X,Y,XE,Y),
        close_up_turns(L).

endmod /* process */ .
```

UTILITIES.PRO

This program contains utility predicates and must be loaded as a module with **PROCESS.PRO** and **BUMPID.PRO**.

```
module utilities.

export          (append / 3, add_item / 3, delete / 3, reverse / 2, singlemember / 2,
        member / 2, stars / 0, first / 2, index1 / 5,
        open / 2, tell / 2, told / 1, get_fm_file / 2, read_file / 0).

/*$eject*/
body.
dynamic (counter / 1).

/*      These file were written by Dave Hutson, originally in C-prolog.      */
/*  It has been converted to run on M-prolog by Robert Powell. Other utility predicates  */
/*  have also been added as needed.                                          */

abs(X,X) :- X >= 0.
abs(X,Y) :- Y is 0 - X.

first([X|L],X).

last([X],X).
last([X|L],Y) :- last(L,Y).

member(X,[X|L]) :- !.
member(X,[Y|L]) :- member(X,L).

delete(X,[],[]).
delete(X,[X|L],M) :- !, delete(X,L,M).
delete(X,[Y|L],[Y|M]) :- delete(X,L,M).

append([],L,L).
append([X|L],L2,[X|L3]) :- append(L,L2,L3).

add_item(X,L,[X|L]):- !.

reverse(L,R) :- reverse2(L,[],R).
reverse2([],L,L) :- !.
reverse2([X|L],R,S) :- reverse2(L,[X|R],S).

/* writes 60 stars */

stars :-
        write("************************************************************"),
         nl,!.
```

```
/*      These predicates are from Professor Rowe's book.          */
/* Iterates repeatedly forward through a predicate until condition  */
/* "done", defined by the user, is satisfied.                       */

iterate(PRED) :-
        repeat, iterate2(PRED), done .

iterate2(PRED) :-
        evaluate(PRED), ! .
iterate2(PRED) .

/*      Written by Mike Bizer for PROCESS.PRO                      */
/* indexes into a list, returning the previous(PP),item(P), and next(PN) */

index1([PP,P,PN|L],N,PP,P,PN) :-
        N = 0,!.
index1([A|L],N,PP,P,PN) :-
        NC is N-1,
        index1(L,NC,PP,P,PN),!.

index(X,[X|L],1) :-
        ! .
index(X,[Y|L],N) :-
        index(X,L,NM1), N is NM1+1 .

/* These utilities were written by Robert Powell.                */

open(CH,OUTFILE):-
        set_channel(outfile(CH),[name=OUTFILE,mode=create]),
        set_output(outfile(CH)),
        told(CH).

tell(CH,OUTFILE):-
        set_channel(outfile(CH),[name=OUTFILE,mode=append]),
        set_output(outfile(CH)).

told(CH) :-
        close_output(outfile(CH)).

get_fm_file(CH,INFILENAME) :-
        set_channel(infile(CH),name=INFILENAME),
        set_input(infile(CH)),
        read_file,
        fail.
get_fm_file(CH,INFILENAME) :-
        close_input(infile(CH)).
```

```
read_file :-
        read_token(file end),!.
read_file :-
        read(X),
        add_statement(X),   /* bottom */
        read_file.

endmod /* utilities */ .
```

BUMPID.PRO

This program identifies the bumps found by **PROCESS.PRO**. It uses the **COORDS.PRO** and **BUMPS.PRO** data files.

```
module bumpid.

/*$eject*/
body.

dynamic (index_counter1 / 1).
dynamic (the_bump / 5).        /* bump fact to be processed */
dynamic (low / 1).
dynamic (left / 1).
dynamic (high / 1).
dynamic (right / 1).
dynamic (bump_coords / 1)    /* coordinates making bump */
dynamic (bump_descr / 3).    /* bump & description list */
dynamic (box / 8).           /* box description         */
dynamic (finish_bump / 1).
dynamic (bumps_to_process / 1).
dynamic (ship_length / 1).

go :-  state(cpu_time,TI),
       add_statement(index_counter1(0)),
       set_state(global_stack,30000),
       set_state(main_stack,10000),
       system(garbage_collection),
       system(compress_stacks),
       set_up_bumpid,iterate(go1),
       state(cpu_time,TF), TOTAL_TIME is TF - TI,
       stars,write("cpu time = "),write(TOTAL_TIME),
       nl.

go1 :-  nl,
       pbump(B1),compute_shape(B1),check_number_id(B1),
       tell(1,"objects"),
       bump_descr(B1,L,O),
       write(B1),nl,
       write('   BUMP_DESCR -->'),
       write(L),nl,
       write('   BUMP_ID --> '),
       write(O),nl,
       told(1),
       clean_corners,index_counter1(X),
       X2 is X + 1, del_statement(index_counter1(X)),
       add_statement(index_counter1(X2)),nl,
       check_statement_table(B1),
       stars.
```

```
check_statement_table(B1) :- state(statement_table,[U,S]),U>300000E0,
        add_statement(finish_bump(b(999,999,999,999))),stars,
        write("Statement table full after bump "),write(B1),stars,!.
check_statement_table(B1).

clean_corners:-
        low(Y17),del_statement(low(Y17)),
        high(Y18),del_statement(high(Y18)),
        right(X18),del_statement(right(X18)),
        left(X17),del_statement(left(X17)),
        bump_coords(XYZ),length(XYZ,N),write(" # segments ==> "),
        write(N),nl,
        del_statement(bump_coords(XYZ)),
        add_statement(bump_coords([])).

done :- finish_bump(B),bconvert(B,X,Y,X1,Y1),
        X=999, X1 = 999.

display_stats :- nl,stars,
        state(main_stack,[U,C]),
        write("main_stack used = "),write(U),nl,
        state(statement_table,[U1,C1]),
        write("statement_table used ="),write(U1),write(" "),write(C1),
        nl.

pbump(B1) :- get_two_bumps(B1,B2),asserta(finish_bump(B2)),
        convertit(B1,X,Y,X1,Y1),convertit(B2,X2,Y2,X3,Y3),
        add_statement(the_bump(B1,X,Y,X1,Y1)),
        add_statement(bump_descr(B1,[],[])),
        create_bump_coords,
        !.

set_up_bumpid :-
        open(1,"objects"),
        tell(1,"objects"),nl,
        write("OBJECTS from BUMPID"),nl,
        told(1),
        bumps(B),add_statement(bumps_to_process(B)),
        start_pt([X,Y]),end_pt([X1,Y1]),
        Z is X1 - X, add_statement(ship_length(Z)),nl,
        add_statement(finish_bump([])),
        add_statement(bump_coords([])),!.

get_two_bumps(B1,B2) :-
        bumps_to_process([B1,B2|B]), write(B1), write(' '), write(B2), nl,
        del_statement(bumps_to_process([B1,B2|B])),
        add_statement(bumps_to_process([B2|B])),!.
```

```
convertit(B,X,Y,X1,Y1) :-
        B=..[b,X,Y,X1,Y1] .
cpconvert(B,X,Y) :-
        B=..[cp,X,Y].
bconvert(B,X,Y,X1,Y1) :-
        B=..[b,X,Y,X1,Y1].

create_bump_coords :- the_bump(B,X,Y,X1,Y1),
        points(L),find_start(X,Y,L,LR),
        create_list(X,Y,X1,Y1,LR),
        del_statement(the_bump(B,X,Y,X1,Y1)),!.
create_bump_coords :- the_bump(B,X,Y,X1,Y1),
        points(L),reverse(L,RL),
        find_rev_start(X1,Y1,RL,LR),
        reverse(LR,RRL),
        create_list(X,Y,X1,Y1,RRL),
        bump_coords([F|C]),
        cpconvert(F,XB,YB),
        cpconvert(D,XB,Y1),append(C,[D],CC),
        add_statement(bump_coords(CC)),
        del_statement(bump_coords([F|C])),
        del_statement(the_bump(B,X,Y,X1,Y1)),
        !.

find_start(X,Y,[A|LR],LR) :-
        A =..[cp,Xp,Yp],X=Xp,Y=Yp,
        bump_coords(LL),append([A],LL,LL1),
        add_statement(bump_coords(LL1)),del_statement(bump_coords(LL)),!.
find_start(X,Y,[A|LR],LR) :-
        cpconvert(A,Xp,Yp),X=Xp,Y<Yp,cpconvert(B,X,Y),not(member(B,[A|LR])),
        bump_coords(LL),
        append([B],LL,LL1),
        add_statement(bump_coords(LL1)),del_statement(bump_coords(LL)),!.
find_start(X,Y,[A|LL],LR) :- find_start(X,Y,LL,LR),!.

find_rev_start(X,Y,[A|LR],LR) :-
        A =..[cp,Xp,Yp],X=Xp,Y=YP,
        bump_coords(LL),append(LL,[A],LL1),
        add_statement(bump_coords(LL1)),del_statement(bump_coords(LL)),!.
find_rev_start(X,Y,[A|LL],LR) :- find_rev_start(X,Y,LL,LR),!.

create_list(X,Y,X1,Y1,[]) :-
        write(" POSSIBLE ERROR, ran out of coordinate points while"),nl,
        write("     processing bump "),write(X),write(" "),write(Y),
        write("    "),write(X1),write(" "),write(Y1),nl,!.
create_list(X,Y,X1,Y1,[A|L]) :-
        A=..[cp,Xf,Yf],Yf < Y1,!.
```

```
create_list(X,Y,X1,Y1,[A|L]) :-
        A=..[cp,Xf,Yf],
        X1 = Xf, Y1 >= Yf, bump_coords(LL),append(LL,[A],LL1),
        add_statement(bump_coords(LL1)),del_statement(bump_coords(LL)),!.
create_list(X,Y,X1,Y1,[A|L]) :-
                                        /* peeling off upper bumps */
        bumps(B),
        cpconvert(A,XT,YT),
        bconvert(TEST,XT,YT,XX,YY),
        member(TEST,B),
        locate_coords(TEST,B,[BUMP_COORDS|LLL]),
        bconvert(BUMP_COORDS,XA,YA,XF,YF),
        bconvert(WORKING_BUMP,X,Y,X1,Y1),
        not(member(BUMP_COORDS,[WORKING_BUMP])), /*ensure not same
                                                        bump*/
        cpconvert(C,XF,YF),
        member(C,[A|L]),
        locate_coords(C,[A|L],[Z|RL]),
        write(" peeling off bump1 "),write(BUMP_COORDS),nl,
        create_list(X,Y,X1,Y1,RL),!.

create_list(X,Y,X1,Y1,[A|L]) :-         /* don't compare with bump using added*/
                                        /* peeling off upper bumps */
        bumps(B),                       /* with created endings   */
        cpconvert(A,XT,YT),
        bconvert(TEST,XT,YT,XX,YY),
        member(TEST,B),
        locate_coords(TEST,B,[BUMP_COORDS|LLL]),
        bconvert(BUMP_COORDS,XA,YA,XF,YF),
        bconvert(WORKING_BUMP,X,Y,X1,Y1),
        not(member(BUMP_COORDS,[WORKING_BUMP])), /*ensure not same
                                                        bump*/
        cpconvert(C,XF,YF),
        locate_fake_coords(X1,Y1,C,[A|L],[Z|RL]),
        write(" peeling off bump2 "),
        write(BUMP_COORDS),nl,
        create_list(X,Y,X1,Y1,RL),!.

create_list(X,Y,X1,Y1,[A|L]) :-
        bump_coords(LL),append(LL,[A],LL1),
        del_statement(bump_coords(LL)),add_statement(bump_coords(LL1)),
        create_list(X,Y,X1,Y1,L),!.
```

```
compute_shape(B1) :- bump_coords([LLOW|L]),
        set_defaults([LLOW|L]),
        check_defaults(B1),
        find_corners(L),
        make_box(XMIN,YMIN,XMAX,YMAX),
        write("##### BOX > "),
        write(XMIN),write(YMIN),write(' '),
        write(XMIN),write(YMAX),write(' '),
        write(XMAX),write(YMAX),write(' '),
        write(XMAX),write(YMIN),nl,
        describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),
        !.

make_box(XMIN,YMIN,XMAX,YMAX) :-
        low(YMIN),
        left(XMIN),
        high(YMAX),
        right(XMAX),
        add_statement(box(XMIN,YMIN,XMIN,YMAX,XMAX,YMAX,XMAX,YMIN)
),!.

find_min(A,B,A) :- A <= B,!.
find_min(A,B,B).
find_max(A,B,A) :- A >= B, !.
find_max(A,B,B).

set_defaults([LLOW|L]) :- LLOW =.. [cp,X,Y],
        add_statement(low(Y)),
        add_statement(left(X)),
        get_last([LLOW|L]),!.

get_last([A|[]]) :- A =..[cp,X,Y],
        add_statement(right(X)),
        add_statement(high(Y)),!.
get_last([A|L]) :- get_last(L).

find_corners([]) :- !.
find_corners([A|L]) :-
        A =.. [cp,XA,YA],high(Y),
        Y < YA,
        add_statement(high(YA)),
        del_statement(high(Y)),find_corners(L),!.



find_corners([A|L]) :-
        A =.. [cp,XA,YA],left(X),
        XA < X,
        add_statement(left(XA)),
        del_statement(left(X)),find_corners(L),!.
```

```
find_corners([A|L]) :-
        A =.. [cp,XA,YA],right(X),
        XA > X,
        add_statement(right(XA)),
        del_statement(right(X)),find_corners(L),!.
find_corners([A|L]):-
        find_corners(L),!.

check_defaults(B1) :-
        bconvert(B1,X,Y,XX,YY),
        check_xmin(X),
        check_ymin(Y),
        check_xmax(XX),
        check_ymax(YY),!.

check_xmin(X) :-
        left(XT),
        XT <= X, !.
check_xmin(X) :-
        left(XT),
        change_left_default(X,XT),!.
check_ymin(Y) :-
        low(YT),
        YT <= Y, !.
check_ymin(Y) :-
        low(YT),
        change_low_default(Y,YT),!.
check_xmax(XX) :-
        right(XT),
        XT >= XX, !.
check_xmax(XX) :-
        right(XT),
        change_right_default(XX,XT),!.
check_ymax(YY) :-
        high(YT),
        YT >= YY, !.
check_ymax(YY) :-
        high(YT),
        change_high_default(YY,YT),!.

change_left_default(NEW,OLD) :-
        add_statement(left(NEW)),
        del_statement(left(OLD)),!.
change_low_default(NEW,OLD) :-
        add_statement(low(NEW)),
        del_statement(low(OLD)),!.
change_right_default(NEW,OLD) :-
        add_statement(right(NEW)),
        del_statement(right(OLD)),!.
```

```
change_high_default(NEW,OLD) :-
        add_statement(high(NEW)),
        del_statement(high(OLD)),!.

/***************** LOCATION descriptors to bump_descr **************/

describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        start_pt([SX,SY]),
        ship_length(X),
        Y is (XMAX + XMIN) / 2,
        Y - SX < 15,
        add_descr(B1,bow),
        describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        start_pt([SX,SY]),
        ship_length(X),
        Y is (XMAX + XMIN) / 2,
        Y  < SX + (X / 3),
        Y - SX >= 15,
        add_descr(B1,forward),
        describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        start_pt([SX,SY]),
        end_pt([EX,EY]),
        ship_length(X),
        Y is (XMAX + XMIN) / 2,
        Y  > SX + (2 * X / 3),
        EX - Y >= 15,
        add_descr(B1,aft),
        describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        start_pt([SX,SY]),
        ship_length(X),
        Y is (XMAX + XMIN) / 2,
        Y < SX + (2 * X / 3),
        Y > SX + (X / 3),
        add_descr(B1,mid_ship),
        describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        end_pt([SX,SY]),
        ship_length(X),
        Y is (XMAX + XMIN) / 2,
        SX - Y < 15,
        add_descr(B1,stern),
        describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) , !.
```

```
/****************** add size descriptors ********************/

describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) :-
        ship_length(X),
        OS is XMAX-XMIN,
        OS/X >= 0.10E0,
        add_descr(B1,extra_large),
        describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) :-
        ship_length(X),
        OS is XMAX-XMIN,
        OS/X >= 0.03E0,
        OS/X < 0.10E0,
        add_descr(B1,large),
        describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) :-
        ship_length(X),
        OS is XMAX-XMIN,
        OS/X < 0.03E0,
        OS/X > 0.015E0,
        add_descr(B1,medium),
        describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) :-
        ship_length(X),
        OS is XMAX-XMIN,
        OS/X < 0.015E0,
        add_descr(B1,small),
        describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) , !.

/***************** add CURVINESS to bump_descr ***************/

describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) :-
        bump_coords(XYZ),length(XYZ,N),N>50,
        add_descr(B1,high_curviness),
        describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) :-
        bump_coords(XYZ),length(XYZ,N),N<15,
        add_descr(B1,no_curviness),
        describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) , !.

/***************** add descriptors to bump_descr ***************/

describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) :-
        in_range(XMAX-XMIN,YMAX-YMIN),
        add_descr(B1,square),
        describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) :-
        XMAX-XMIN > YMAX-YMIN + 5,
        add_descr(B1,flat_rectangle),
        describe_shape(B1,[LLOWIL],XMIN,YMIN,XMAX,YMAX) , !.
```

```
describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        XMAX-XMIN + 5 < YMAX-YMIN,
        add_descr(B1,tall_rectangle),
        describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        XMAX-XMIN <= 5, (XMAX-XMIN)/(YMAX-YMIN)<=0.5E0,
        add_descr(B1,pole),
        describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        (XMAX-XMIN)/(YMAX-YMIN)<=0.17E0,
        bump_coords(XYZ),length(XYZ,N),N<30,
        add_descr(B1,pole),
        describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) , !.
describe_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :- !.

add_descr(B1,X) :-
        bump_descr(B1,DL,O),not(member(X,DL)),
        append([X],DL,LL),add_statement(bump_descr(B1,LL,O)),
        del_statement(bump_descr(B1,DL,O)).

/********************** IDENTIFY SHAPE ****************************/

identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        bump_descr(B1,DL,O),member(pole,DL),
        not(member(bow,DL)),not(member(stern,DL)),
        add_id(B1,antenna),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),!.

identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        bump_descr(B1,DL,O),member(pole,DL),
        not(member(mid_ship,DL)),not(member(forward,DL)),
        not(member(aft,DL)),
        add_id(B1,mast/support),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),!.

identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        bump_descr(B1,DL,O),member(flat_rectangle,DL),
        not(member(small,DL)),not(member(extra_large,DL)),
        member(high_curviness,DL),
        not(member(mid_ship,DL)),
        add_id(B1,gun-turret/weapon-system),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),!.
```

```
identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        bump_descr(B1,DL,O),member(flat_rectangle,DL),
        not(member(mid_ship,DL)), not(member(small,DL)),
        not(member(extra_large,DL)),
        not(member(high_curviness,DL)),
        not(member(no_curviness,DL)),
        add_id(B1,weapon-system),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),!.

identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        bump_descr(B1,DL,O),member(flat_rectangle,DL),
        member(forward,DL), not(member(small,DL)),
        (member(no_curviness,DL)),
        add_id(B1,superstructure),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),!.

identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        bump_descr(B1,DL,O),
        member(mid_ship,DL),
        not(member(pole,DL)),not(member(small,DL)),
        not(member(tall_rectangle,DL)),
        (XMAX-XMIN)/(YMAX-YMIN) <= 2.0E0,
        (XMAX-XMIN)/(YMAX-YMIN) >= 0.5E0,
        add_id(B1,radar),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),!.

identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        bump_descr(B1,DL,O),
        member(mid_ship,DL), not(member(no_curviness,DL)),
        not(member(pole,DL)),not(member(square,DL)),
        not(member(tall_rectangle,DL)),
        (XMAX-XMIN)/(YMAX-YMIN) <= 2.0E0,
        (XMAX-XMIN)/(YMAX-YMIN) >= 0.5E0,
        add_id(B1,radar),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),!.

identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        bump_descr(B1,DL,O),member(tall_rectangle,DL),
        not(member(pole,DL)),not(member(large,DL)),
        (XMAX-XMIN)/(YMAX-YMIN) <= 2.0E0,
        add_id(B1,mast/support),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),!.

identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        bump_descr(B1,DL,O),not(member(pole,DL)),
        not(member(small,DL)),not(member(high_curviness,DL)),
        not(member(forward,DL)),
        add_id(B1,superstructure),
        identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX),!.
```

```
identify_shape(B1,[LLOW|L],XMIN,YMIN,XMAX,YMAX) :-
        check_if_empty(B1),!.

add_id(B1,X) :-
        bump_descr(B1,DL,O),not(member(X,O)),
        append([X],O,OL),add_statement(bump_descr(B1,DL,OL)),
        del_statement(bump_descr(B1,DL,O)).

/********** Change list to OR if multiple identifications made ****/

check_number_id(B1) :-
        bump_descr(B1,DL,O),length(O,N),write(N),N>1,
        write(O),nl,change_to_or(O,or,OO),
        add_statement(bump_descr(B1,DL,OO)),
        del_statement(bump_descr(B1,DL,O)),!.
check_number_id(B1) :- !.

change_to_or([A,B|L],OR,LL) :-
        append([OR],[A],Y),append([B],Y,LL),!.

check_if_empty(B1) :-
        bump_descr(B1,DL,[]),
        add_id(B1,unknown),!.
check_if_empty(B1).

/*****************************************************************/
locate_coords(COORD,[A|L],[A|L]) :-
        first_one(COORD,[A|L]),!.
locate_coords(COORD,[A|L],[A|L]) :-
        cpconvert(COORD,X,Y),
        cpconvert(A,XT,YT),
        X=XT, Y > YT,!.
locate_coords(COORD,[A|L],LL) :-
        locate_coords(COORD,L,LL).

locate_fake_coords(X1,Y1,COORD,[A|L],[A|L]) :-
        cpconvert(COORD,XF,YF),cpconvert(A,XT,YT),
        XT=XF,YT<YF,
        bump_coords(BL), append(BL,[COORD],BLL),append(BLL,[A],LL),
        add_statement(bump_coords(LL)),
        del_statement(bump_coords(BL)),
        !.
locate_fake_coords(X1,Y1,COORD,[A|L],LL) :-
        locate_fake_coords(X1,Y1,COORD,L,LL).

first_one(X,[X|L]).
```

```
/************************ utilities ******************************/
in_range(VAR1,VAR2) :-
        VAR1 <= VAR2 + 3,
        VAR1 >= VAR2 - 3.

endmod /* bumpid */ .
```

APPENDIX C - OUTPUT FROM GARCIA (LAMPS)

GAR-OBJ OUTPUT

This is an example of output from the **GARCIA (LAMPS)** silhouette, showing the bump locations, description, and identification.

```
OBJECTS from BUMPID
b(341,484,361,484)
  BUMP_DESCR -->[flat_rectangle,high_curviness,large,forward]
  BUMP_ID --> [gun-turret/weapon-system]
b(340,476,364,476)
  BUMP_DESCR -->[flat_rectangle,no_curviness,large,forward]
  BUMP_ID --> [superstructure]
b(407,485,415,485)
  BUMP_DESCR -->[flat_rectangle,large,forward]
  BUMP_ID --> [weapon-system]
b(403,478,416,478)
  BUMP_DESCR -->[flat_rectangle,no_curviness,medium,forward]
  BUMP_ID --> [superstructure]
b(457,537,459,537)
  BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,mid_ship]
  BUMP_ID --> [antenna]
b(456,522,457,522)
  BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,mid_ship]
  BUMP_ID --> [antenna]
b(465,564,465,564)
  BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,mid_ship]
  BUMP_ID --> [antenna]
b(465,522,467,522)
  BUMP_DESCR -->[pole,tall_rectangle,small,mid_ship]
  BUMP_ID --> [antenna]
b(489,526,499,526)
  BUMP_DESCR -->[square,no_curviness,small,mid_ship]
  BUMP_ID --> [unknown]
b(533,543,536,543)
  BUMP_DESCR -->[flat_rectangle,high_curviness,large,mid_ship]
  BUMP_ID --> [radar]
b(536,571,541,571)
  BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,mid_ship]
  BUMP_ID --> [antenna]
b(559,618,561,618)
  BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,mid_ship]
  BUMP_ID --> [antenna]
```

b(565,618,569,618)
BUMP_DESCR -->[square,no_curviness,small,mid_ship]
BUMP_ID --> [unknown]
b(560,601,561,601)
BUMP_DESCR -->[tall_rectangle,small,mid_ship]
BUMP_ID --> [mast/support]
b(549,592,563,592)
BUMP_DESCR -->[flat_rectangle,large,mid_ship]
BUMP_ID --> [superstructure]
b(529,539,553,539)
BUMP_DESCR -->[flat_rectangle,no_curviness,large,mid_ship]
BUMP_ID --> [superstructure]
b(552,544,553,544)
BUMP_DESCR -->[tall_rectangle,medium,mid_ship]
BUMP_ID --> [mast/support,or,superstructure]
b(551,571,553,571)
BUMP_DESCR -->[square,medium,mid_ship]
BUMP_ID --> [radar,or,superstructure]
b(527,528,554,528)
BUMP_DESCR -->[flat_rectangle,large,mid_ship]
BUMP_ID --> [radar,or,superstructure]
b(485,514,559,514)
BUMP_DESCR -->[flat_rectangle,extra_large,mid_ship]
BUMP_ID --> [superstructure]
b(587,507,589,507)
BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,mid_ship]
BUMP_ID --> [antenna]
b(602,502,617,502)
BUMP_DESCR -->[flat_rectangle,no_curviness,medium,mid_ship]
BUMP_ID --> [superstructure]
b(631,516,633,516)
BUMP_DESCR -->[pole,tall_rectangle,small,mid_ship]
BUMP_ID --> [antenna]
b(626,502,629,502)
BUMP_DESCR -->[tall_rectangle,small,mid_ship]
BUMP_ID --> [mast/support]
b(651,494,653,494)
BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,mid_ship]
BUMP_ID --> [antenna]
b(675,496,699,496)
BUMP_DESCR -->[flat_rectangle,high_curviness,large,aft]
BUMP_ID --> [gun-turret/weapon-system]
b(737,515,739,515)
BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,aft]
BUMP_ID --> [antenna]
b(745,520,747,520)
BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,aft]
BUMP_ID --> [antenna]
b(745,515,755,515)
BUMP_DESCR -->[tall_rectangle,medium,aft]
BUMP_ID --> [mast/support,or,superstructure]

b(788,517,798,517)
  BUMP_DESCR -->[no_curviness,small,aft]
  BUMP_ID --> [unknown]
b(763,515,802,515)
  BUMP_DESCR -->[flat_rectangle,no_curviness,large,aft]
  BUMP_ID --> [superstructure]
b(745,514,803,514)
  BUMP_DESCR -->[square,large,aft]
  BUMP_ID --> [superstructure]
b(675,494,803,494)
  BUMP_DESCR -->[flat_rectangle,high_curviness,extra_large,aft]
  BUMP_ID --> [unknown]
b(725,499,803,499)
  BUMP_DESCR -->[flat_rectangle,extra_large,aft]
  BUMP_ID --> [superstructure]
b(932,504,933,504)
  BUMP_DESCR -->[pole,no_curviness,small,stern]
  BUMP_ID --> [mast/support]
b(931,497,931,497)
  BUMP_DESCR -->[pole,no_curviness,small,stern]
  BUMP_ID --> [mast/support]
b(929,491,934,491)
  BUMP_DESCR -->[pole,tall_rectangle,no_curviness,small,stern]
  BUMP_ID --> [mast/support]

LIST OF REFERENCES

1. Bernier, Denise R., *An Intelligent Computer-Aided Instruction System for Naval Ship Recognition,* Master's Thesis, Naval Postgraduate School, Monterey, California, June 1989.

2. Ballard, Dana H. and Brown, Christopher M., *Computer Vision,* Prentice-Hall, 1982.

3. Cox, K.C., Roman, G., Ball, W.E., and Laine, A. F., "Rapid Search for Spherical Objects in Aerial Photographs," *IEEE Proceedings: Computer Vision and Pattern Recognition, 1988*, 1988.

4. Grogan, Timothy Alan and Mitchell, O. Robert, *Shape Recognition and Description: A Comparative Study*, Purdue University Lafayette in School of Electrical Engineering, 1982.

5. Todd, Henry S., "A Descriptive Pattern Recognition System Applied to Pictorial Patterns Where the Discriminating Information is Carried in the Object Shape," *IEEE Proceedings: Computer Vision and Pattern Recognition, 1988*, pp. 430-436, 1988.

6. Opsahl, Torstein, "Automated Target Detection," *Fifth Annual Intelligence Community AI Symposium,* Defense Intelligence College, 1987.

7. Telephone conversation between Mike Hord, MRJ, Inc., and the author, 8 June 1989.

8. McKeown, D. M., and Harvey, W. A., "Rule Based Interpretation of Aerial Imagery," *IEEE Conference on Computer Vision,1985*, pp. 570-585, 1985.

9. McKeown, D. M., Harvey, W. A., and Wixson, L. E., "Automating Knowledge Acquisition for Aerial Image Interpretation," *Computer Vision, Graphics, and Image Processing,* 46, pp. 37-81, April 1989.

10. Niblack, W., Petkovic, D., and Damian, D., "Experiments and Evaluations of Rule Based Methods in Image Analysis," *IEEE Proceedings: Computer Vision and Pattern Recognition, 1988*, pp. 123-128, 1988.

11. Smyrniotis, Chuck and Dutta, Kalyan, "A Knowledge-Based System for Recognizing Man-Made Objects in Aerial Images," *IEEE Proceedings: Computer Vision and Pattern Recognition, 1988*, pp. 111-117, 1988.

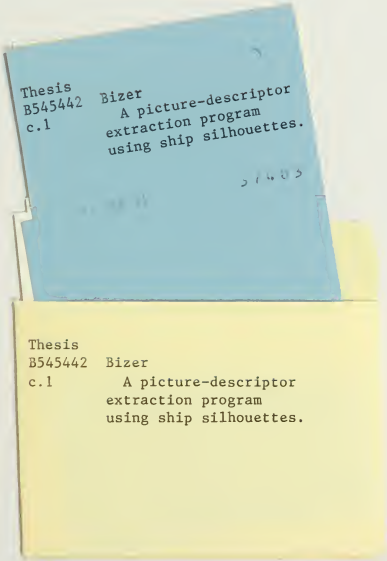
12, *Jane's All the Worlds Fighting Ships 1985-86,* Jane's Publishing Inc., pp. 216-217, 1986.

13. Pavlidis, Theo, *Algorithms for Graphics and Image Processing,* Computer Science Press, 1987.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Chief of Naval Operations Director, Information Systems (OP-945) Navy Department Washington, D.C. 20350-2000	1
4.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943-5000	2
5.	Curricular Officer, Code 37 Computer Technology Naval Postgraduate School Monterey, California 93943-5000	1
6.	Associate Professor Neil C. Rowe, Code 52Rp Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	2
7.	Associate Professor Michael J. Zyda, Code 52Zk Computer Science Department Naval Postgraduate School Monterey, California 93943-5000	1

8.	Dr. Hank Smith Education Coordinator Patrol Squadron THIRTY-ONE Naval Air Station Moffett Field, California 94035	1
9.	CPT Michael J. Bizer 1204 Catskill Circle Huntsville, Alabama 35802	2


Thesis
B545442
c.1
Bizer
A picture-descriptor
extraction program
using ship silhouettes.
Thesis
B545442
c.1
Bizer
A picture-descriptor
extraction program
using ship silhouettes.